TEC | Tecnológico de Costa Rica

Tercer Proyecto: Compresión con árboles de Huffman

Curso: IC-1803 Taller de programación

Estudiantes:
Marco Álvarez Grijalba
Carnet: 2022939167
Marco Rodriguez Vargas
Carnet: 2022149445
Yarman Charpentier Castellón
Carnet: 2022363082

Profesor: Edgar Rojas Muñoz

Entrega: 19 de Junio del 2022

1. Resumen Ejecutivo

El proyecto consiste en desarrollar exitosamente un programa capaz de comprimir y descomprimir archivos por medio del algoritmo de Huffman. El algoritmo está basado en el cálculo de frecuencias de los bytes que aparecen en el archivo, para luego realizar un árbol binario con el cual se busca "balancear" dichas frecuencias. Dicho arbol es utilizado para crear un archivo el cual contiene toda la información obtenida al aplicar el proceso de compresión, en general estos archivos (con extenxion $.huff$) ocupan menos espacio en la memoria que los archivos originales, aunque en algunos casos pueden llegar a ocupar el mismo espacio, sin embargo, nunca llegarán a ocupar más espacio que los archivos originales.La programación de árboles binarios, listas y recursión programados en python 3 por medio del manejo y la compresión de archivos. Los árboles de Huffman como tal son una manera en la que se puede comprimir archivos en los que no exista un extravío de información; el producto original puede reconstruirse partiendo del archivo comprimido por lo que su resultado será el mismo del principio.

Para los archivos de texto se tomaron como posibles bytes todos los 256 simbolos alfanumericos posibles.Para el presente proyecto se desarollaron dos programas *.py* uno llamado *compresor* y *descompresor* El primero recibe el archivo que se desea comprimir y escribe tres archivos, uno con la versión comprimida del archivo, uno con la tabla de códigos y otro con algunas estadísticas del árbol de huffman. El segundo programa recibe los dos primeros archivos y con ellos "descomprime.[el] archivo, obteniendo finalmente el archivo original sin errores. El algoritmo de Huffman se desarrollo de utilizando una funcion iterativa para encontrar las frecuencias y crear el arbol con los bytes encontrados y sus frecuencias, para almacenarlo en un archivo $.huff$ con el mismo nombre del archivo original.

Para la descompresion del archivo se utilizo una funcion iterativa que lee la información del archivo $.huff$ para reconstruir el archivo original. Debido a que algunos simbolos alfanumericos se representaban con un unico bit en vez de un byte completo se opto por iguala la cantidad de bits para cada simbolo, rellenando los bits faltantes con 0 a la izquierda hasta obtener los 8 bite que componen un byte. La solución se estará evaluando mediante los tiempos de compresión y descompresión de los archivos de diferentes tamaños para posteriormente ser comparados.

Los resultados muestran que aquellosa archivos de menor tamaño tendran menor duración, en cuanto a tipos de archivos resultó ser mas efectivo en archivos text reduciendo drásticamente su tamaño pero en archivos de tipo pdf o jpg su tamaño se vio reducido de forma mínima mientras que los archivos que estén mayormente comprimidos anterior al compresor tendrán menor tiempo comparado a aquellos archivos que no estén comprimidos. La mayor ventaja que presentan los archivos comprimidos es el ahorro y reducción del espacio que se almacena en el disco duro, esto significa que la posibilidad de agregar más archivos. Ádemas esto signifca que estos archivos comprimidos tienen mayor compatibilidad ya que al reducir su peso son más accesibles para su distribución.

2. Introducción

En la computacíon un código Huffman es el que se usa de manera común para la compresión de archivos sin que se pierda algún dato en el proceso de compresión o descompresión. El algoritmo fue desarrollado por David Huffman cuando fue Doctor en Ciencias del Instituto Tecnológico de Massachusetts y su publicación ocurrió en el año mil novecientos cincuenta y dos con el nombre de: "Un método para la construcción de códigos de redundancia mínima".

El proyecto consiste en la programación de árboles binarios, listas y recursión programados en python 3 por medio del manejo y la compresión de archivos. Los árboles de Huffman como tal son una manera en la que se puede comprimir archivos en los que no exista un extravío de información; el producto original puede reconstruirse partiendo del archivo comprimido por lo que su resultado será el mismo del principio.

Por lo tanto para lo mencionado anteriormente se escribiran dos programas, compresor y descompresor, el primero se encargará de un sólo parámetro, el nombre del archivo que se necesite comprimir y este deberá escribir tres archivos, el primero será la versión comprimida del archivo, el segundo con la tabla de códigos y el último con algunas estadísticas correspondientes al árbol de huffman. Por otro lado, el segundo programa, el descompresor, será el que debe recibir los dos primeros archivos creados por el compresor y se encargará de descomprimir el archivo, de manera que este quede idéntico al archivo que se utilizó originalmente.

Índice

3. Marco teórico

El lenguaje de programacion usado para este proyecto fue Python, el cual es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo, Python fue creado a finales de los años ochenta por Guido van Rossum en el Centro para las Matemáticas y la Informática, en los Países Bajos, capaz de manejar excepciones e interactuar con el sistema operativo Amoeba. El nombre del lenguaje proviene de la afición de su creador por los humoristas británicos Monty Python. Este lenguaje es un sucesor del lenguaje ABC, el cual es un lenguaje de programacion imperativo de proposito general desarrollado tambien en el Centro para las Matemáticas y la Informática. la primera version de Python fue publicada el 20 de febrero de 1991.
El proyecto se desarrollo en PyCharm, el cual es una IDE (entorno de desarrollo integrado, por sus siglas en ingles) especificamente desarrollado para python por la compañía checa JetBrains en el 2011, la cual provee analisis de codigo, un debugger grafico, un tester integrado y soporte web Para este proyecto se hizo uso de la bibliotecas *math* la cual da acceso a funciones matematicas definidas por el estandar de C (otro lenguaje de programacion), con la excepción de que estas no pueden ser usadas con números complejos.

Bytes: En la computación es la unidad de información base, está conformada por 8 bits.

Bits: En la computacion, es la unidad mínima de información. Es un dígito en el sistema de numeración binario, pudiendo tomar un valor de 1 o 0.

Compresión y descompresión de archivos: La compresión de archivos es un proceso en el cual se logra, por medio de un algoritmo, que un archivo ocupe menos espacio en la memoria de un computador. Existen 2 metodos principales denominados *lossy* y *lossless*, sin embargo, para este proyecto se desarrollo un algoritmo *lossless*. Este tipo de algoritmo permite la compresión de archivos sin perder ningún tipo de información en el proceso, por medio de la busqueda e identificación de repeticiones y patrones en los archivos, guardando una sola vez los datos repetidos con instrucciones de como volverlos a su estado inicial, este proceso es el denominado descmpresión.

Árbol de Huffman: Se les llama arboles de Huffman a los arboles binarios obtenidos al aplicar el algoritmo de Huffman, los cuales almancenan la información de un archivo comprimido.

Algoritmo de Huffman: Es un algoritmo que calcula la frecuencia en que bytes determinados aparecen en un archivo, seguidamente, genera un arbol binario en el que balancea dichas frecuencias obtenidas, con este arbol se obtendra toda la información necesaria para la compresion y descompresión del archivo necesitado.

4. Descripción de la solución

4.1. Compresión

Se nos da un archivo cualquiera. Para el cual necesitamos leer todos sus bytes, esto lo logramos con un readbyte de python. Posteriormente calculamos la frecuencia de cada byte, para lo cual creamos una lista con 256 posiciones, a la cual le sumaremos 1 cada que un byte aparezca en el archivo. La razón por la cual es una lista vacia de 256 posiciones, es para que al buscar la frecuencia de un byte, el valor del byte coincida con su indice.

Ejemplo: Lista vacia = [Posicion 0: 50]

listavacia[byte 0] Esto de arriba devolvería 50. Lo cual hace el buscar el valor muy eficiente.

Seguimos con crear una lista de bytes que si aparecen, en esa lista tendremos que cada elemento coincide de: [Frecuencia byte, byte]

Luego ordenamos esta lista para que los elementos mas pequeños esten al final. Procedemos con la creación del arbol, obtenemos el hijo izquierdo y el derecho con un pop. Como ya esta ordenado, sabemos que va a cumplicar con las propiedades del arbol binario cuando lo agreguemos. Despues convertimos los hijos en un nodo, donde el elemento 0 es la suma de las frecuencias de bytes, y los hijos son o un byte y su frecuencia, u otro nodo. Al final obtendremos un arbol cuyas raices son bytes y frecuencias, y todos sus nodos son sumas de frecuencias. Las frecuencias no las necesitamos despues, pero con esta forma del arbol obtenemos los codigos, donde los bytes mas frecuentes tendran un código mas sencillo, para poder ahorrar espacio.

Luego conseguimos las rutas a las hojas, las cuales son los códigos. Hacemos una lista con los codigos, donde el indice es el byte que corresponde a cada código.

Procedemos con la compresión, creamos un huff. Leemos los bytes del archivo original, y cada vez que aparezca un byte, escribimos su codigo en un string, hasta que este sea un string de 8 caracteres que representan los bits del byte. Al final si queda un byte incompleto lo guardamos en el .table.

Tambien hacemos el .stats, guardamos la extensión del archivo para cuando lo descomprimamos.

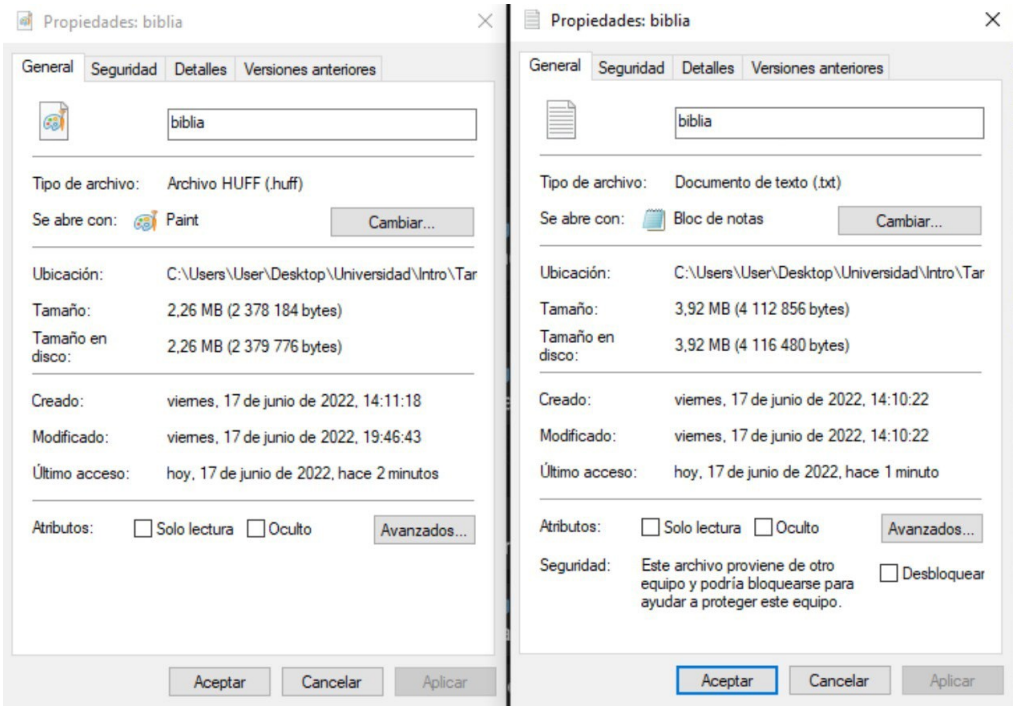
Eso es todo por la parte de compresión.

4.2. Descompresión

Con el archivo .huff y el .table procedemos con la descompresión. Leemos la tabla y hacemos una lista con ella. Al final de la tabla esta el último byte y la extensión del archivo. Con los códigos rearmamos el arbol, cada código es una instrucción de derecha o izquierda, iremos creando nuestro cámino ya que nunca se dara el caso de que el cámino entero ya esté hecho. Como mínimo crearemos la hoja. Leemos los bytes del archivo y los pasamos a binario con la función bin. Le agregamos los 0s necesarios porque bin nos da la expresión mas simple del byte. Los bytes del archivo en binario son instrucciones para el recorrido del arbol, por cada byte recorremos el arbol donde 1 es ir a la derecha, 0 es ir a la izquierda. Cuando encontramos una hoja, escribimos el byte y nos devolvemos a la raíz. Al final lo haremos con el byte final. Y la descompresión estaría finalizada.

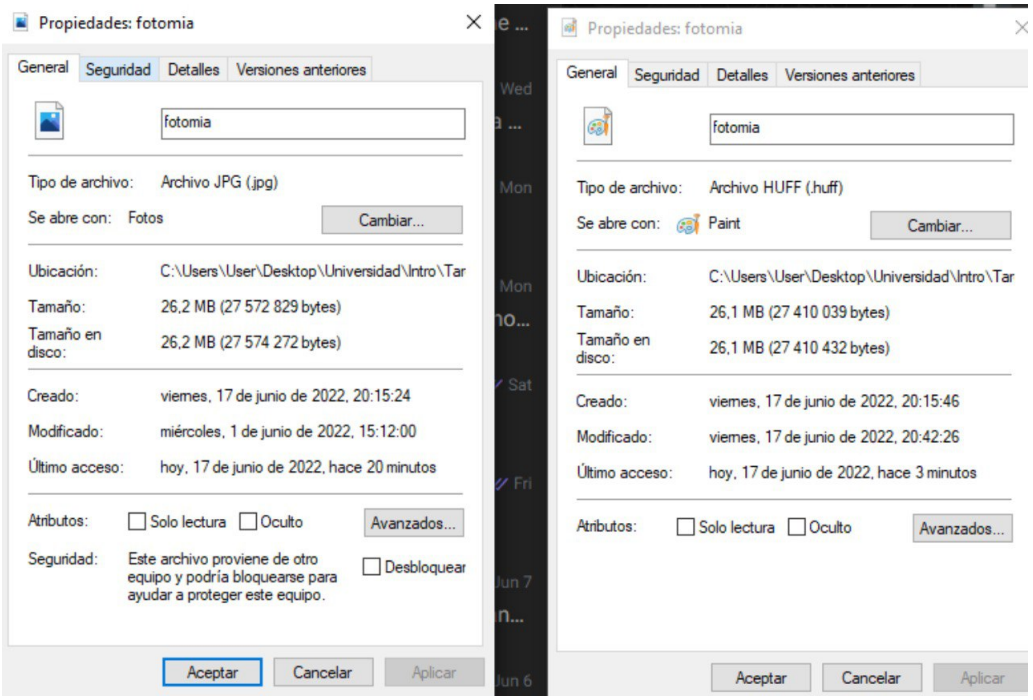
5. Resultados de las pruebas

A continuacion, se presentarán diferentes resultados de pruebas hechas con los programas desarrollados:



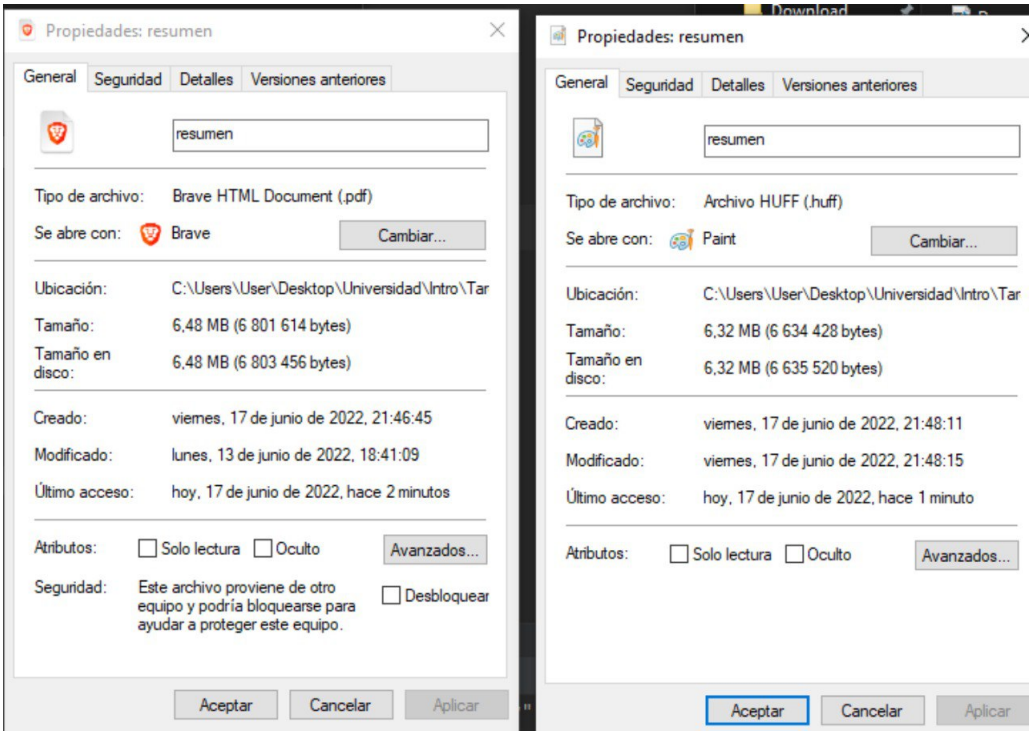
A la derecha se puede apreciar el archivo original, el cual ocupa un espacio de 4 111 856 bytes, mientras que a la izquierda se puede apreciar el archivo ya comprimido, que ocupa 2 378 184, poco más de la mitad mitad que el archivo original.

Para esta prueba el programa de compresion duró cerca de 2 segundos para comprimir el archivo, y cerca de 7 segundos para descomprimirlo.

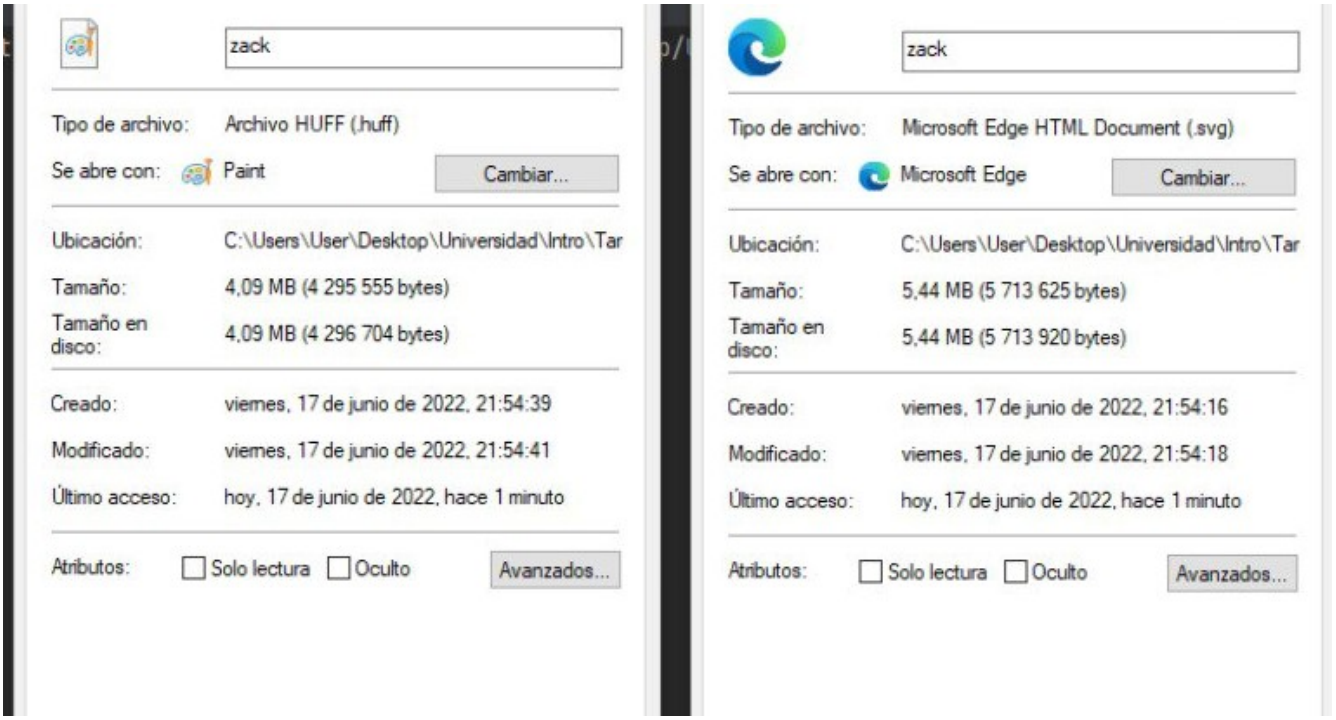


En este caso se comprimio un archivo JPG, izquierda se puede apreciar el archivo original, el cual ocupa un espacio de 27 572 829 bytes, mientras que a la derecha se puede apreciar el archivo ya comprimido, que ocupa 27 410 039, en este caso se redujo muuy poco el tamaño ocupado por el archivo.

En este caso, el programa de compresion duró cerca de 17 segundos para comprimir el archivo, y cerca de 73 segundos, o un minuto y 13 segundos, para descomprimirlo.



En este caso, se comprimio un archivo PDF, en este se puede apreciar que la diferencia entre la memoria ocupada entre el archivo original y el archivo comprimido es mínima. En este caso el tiempo de compresión fue de alrededor de 4 segundos y el tiempo de descompresión fue de cerca de 18 segundos



Para este caso se tomó un archivo .svg, el cual inicialmente ocupaba un espacio de 5 713 625 bytes, mientras que el archivo comprimido ocupa 4 295 555 bytes, teniendo una diferencia considerable entre ambos. Para esta prueba el programa de compresión tuvo una duración de alrededor de 3 segundos, mientras que para su descompresion se requirieron cerca de 12 segundos

6. Conclusiones

Se llega a la conclusión de que la compresión y descompresión de archivos fue la esperada y que el código se llega a ejecutar de manera correcta para que los archivos sean comprimidos y descomprimidos sin afecter sus bytes. Ádemas se llega a la conclusión de que la compresión de archivos resulta útil en aquellos casos en los que se ocupe liberar memoria o que se ocupe guardar varios archivos en una memoria ocupando menos espacio; esto gracias a que fueron comprimidos posteriormente.

De lo mencionado anteriormente se puede decir que se llega a demostrar la utilidad de los arboles de huffman por medio de los casos de prueba en los que se muestra que los archivos disminuyen su tamaño por lo que se le puede dar un sin fin de usos al compresor y en el momento que se necesite el archivo se puede emplear el descompresor para la respectiva descompresión del archivo y de esta manera tener el archivo que se tenia originalmente sin cambios.

Por lo que podemos decir que el código ejecuta de manera correcta el compresor y el descompresor al no haber una alteración de bytes una vez que el archivo pase por el compresor y posteriormente el descompresor. Este era el resultado esperado para el proyecto y fue el obtenido en los casos de prueba mencionados en la sección pasada. Se llega a pensar que la compresión de archivos resulta una manera en la que se puede optimizar el almacenamiento de una computadora con ayuda de lo aprendido a lo largo del curso.

7. Aprendizajes

Yarman Charpentier : Aprendí mucho de manejo de bits y bytes,como se relación y como los interpreta una computadora. Mejore mucho mi práctica en manejo de arboles binarios, su recorrido, creación y recreación. Y sobre todo considero que me se comunicar mejor con un computador gracias a este proyecto.

Marco Álvarez: Logré comprender mejor la importancia de la compresionde archivos. Esta puede llegar a ser muy útil para ahorrar espacio dentro de lo computadores, estos archivos comprimidos tambien permiten una mayor facilidad a la hora de ordenar y buscar estos archivos, ademas para la creación de los dos programas, al requerir del uso del arboles y todo lo relacionado con ellos, pude comprender más acerca de sus utilidad y funcionamiento

Marco Rodriguez: Este proyecto me ayudó a mejorar mi entendimiento de los arboles y su funcionamiento. En este proyecto tuve la oportunidad de practicar otros conceptos vistos en el pasado como la recursividad y las listas. Ádemas tuve que investigar mas sobre el tema para entender mejor el por qué se utiliza y su importancia.

8. Bibliografía

[1] I. Challenger-Pérez, Y. Díaz-Ricardo and R.A. Becerra-García, "El lenguaje de programación Python", Ciencias Holguín, vol. 20, no. 2, pp. 1-13.

[2] Ramirez, E.(2017). Fundamentos de Programación. Sin editorial

[3] Van Leeuwen, Jan (1976). .ºn the construction of Huffman trees"(PDF). ICALP: 382–410. Retrieved 2014-02-20.